Subject: Re: Global variables and IDL
Posted by steinhh on Wed, 21 Apr 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <371CD72E.C77A38A7@io.harvard.edu>
Martin Schultz <mgs@io.harvard.edu> writes:
[...]
> But here are two examples where I used common blocks --
> and I would be happy
> to learn how I could have avoided them:
>
> * in my EXPLORE tool (which can handle several "instances" at
> least if opened from within), I use a common block to keep track
> of the drawing windows that have been opened and used. This is
> needed to kill a window when the associated widget is closed as
> well as to open a new window for a new widget instance. At first
> it would seem that I could simply use the /free keyword to
> WINDOW and store the window number locally with the widget, but
> I have to close *all* windows when I exit the program. Should I
> use the event notification method? Sounds like a viable thing --
> but I would have to rewrite a large part of a running program
> which my boss never likes ...

I've no experience with the EXPLORE tool, but from your description here it seems like you're creating normal draw windows that "belong" to a given widget instance. Why not simply create a detached widget_base containing a widget_draw instead, then get the draw window number (for WSET,WIN) through WIDGET_CONTROL, DRAW_ID,GET_VALUE=WIN, and store it in the info structure for the "owner" widget instance. By setting the owner widget instance as the group leader for the detached top level base containing the widget draw window, the window will automatically be killed whenever your widget instance is killed. You should take a look at some of David F's applications to ensure that resizing the draw windows will work, though...

> * in our new 3D model output analysis tool, we store data
> descriptors for all data that has been read in a common
> block. This makes these data available to all instances of the
> tool (although there currently is only one and it is not even
> widgetized)

And non-widgetized suites of programs are in fact a lot harder to do without any common blocks at all. Nothing will do except explicitly passing (a pointer to) data around in all function calls. Clean as h*ll, but not very practical!

But it's ok (in my humble opinion) to use common blocks to

implement a system to keep track of such "global" data. I.e.,
make two-three routines that share one (private) common block,
along the lines of:

```
  REGISTER_ITEM,"NAME",DATA       ;; Will "undefine" DATA,
                          ;; to avoid copying
  DATAPTR = RETRIEVE_ITEM("NAME") ;; Returns a pointer to
                          ;; the registered DATA item.
```

If you want to avoid pointer notation, you could
temporarily put your data into local variables by

```
  DATA = TEMPORARY(*DATAPTR) ;; Get data without copying
  ;; Do the processing
  *DATAPTR = TEMPORARY(DATA) ;; Put it back.
```

To be crash tolerant and still avoid the pointer notation as well
as data copying, you'd need a slightly different approach, with
one publicly available common block (yes!!!), and do something
like this in your "client" programs:

```
PRO MY_ROUTINE
  COMMON DATA_SYSTEM_CACHE,ITEMNAME,DATA ;;You can use whatever
                              ;;variable names that
                              ;;are appropriate for
                              ;;this procedure.

  NEWDATA = READ_DATA(FILENAME)
  REGISTER_ITEM,"NAME",NEWDATA

  NEWDATA = READ_DATA(FILENAME2)
  REGISTER_ITEM,"ANOTHER_NAME",NEWDATA

  CHECKIN_ITEM,"NAME"  ;; Stores any current DATA_SYSTEM_CACHE
                ;; contents and puts the requested
                ;; data there instead.
  DATA.element = 5.0

  CHECKIN_ITEM,"ANOTHER_NAME" ;; Ditto.
  DATA.something_else = "True"

END
```

I made something like this (though a *lot* less general) back in,
oh, 1994 in fact, and the software is still in operation. Of
course back then we didn't even have handles, much less
pointers. I had to use unrealized widget bases, and the /no_copy
keyword...

Regards,

Stein Vidar